Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Smrkolj Koželj

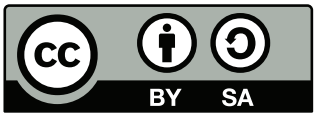
Interaktivna vizualizacija prostorskih podatkov v Unity3D

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana, 2017



Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Interaktivna vizualizacija prostorskih podatkov v Unity3D

Tematika naloge:

V diplomski nalogi implementirajte interaktivno vizualizacijo 3D terena Slovenije, ki bo združevala LiDAR podatke z ortofoto posnetki. Pri tem poskrbite za ustrezno predstavitev in poravnavo teh podatkov, implementirajte sistem ločljivostnih nivojev in zagotovite dobro uporabniško izkušnjo s paralelnim nalaganjem in obdelavo podatkov. Vizualizacija naj bo implementirana v okolju Unity3D.

Zahvaljujem se doc. dr. Matiji Maroltu za mentorstvo pri diplomski nalogi. Zahvaljujem se asistentu dr. Cirilu Bohaku za njegovo pomoč in nasvete pri izdelavi diplomske naloge. Zahvaljujem se svoji družini in prijateljem, brez katerih ne bi mogel priti do sem in kateri so me podpirali skozi študij. Posebej bi se zahvalil prijatelju in kolegu Žiga Simončiču za sodelovanje in pomoč tekom študija.

Kazalo

Slike

Povzetek

Naslov: Interaktivna vizualizacija prostorskih podatkov v Unity3D

Avtor: Nejc Smrkolj Koželj

V diplomski nalogi smo implementirali prototip interaktivne vizualizacije 3D terena Slovenije. Skupaj smo povezali LiDAR višinske podatke v obliki višinskih slik in ortofoto posnetke, ki smo jih pridobili s strežnikov Agencije Republike Slovenije za okolje. Prototip smo implementirali v pogonu Unity3D s programskim jezikom C#.

Za uspešno interaktivno vizualizacijo smo implementirali več podpornih sistemov. Teren Slovenije smo razdelili na več plošč, pri čemer je vsaka plošča sestavljena iz metapodatkov in terena. Teren generiramo iz višinskih slik in ortofoto posnetkov. Za poravnavo višinskih slik in ortofoto posnetkov smo implementirali metodo za poravnavo z odmikom in sistem kontrolnih točk.

Zaradi velike količine plošč smo implementirali sistem ločljivostnih nivojev. Bližje kot je kamera terenu, natančnejše plošče uporabimo. Poleg tega s sistemom ločljivostnih nivojev bolje izrabljamo pomnilnik, saj je naenkrat naloženih v pomnilniku manj plošč. Plošče smo povezali v hierarhijo, za kar smo implementirali lastno podatkovno strukturo TreeStructure, ki združuje plošče v mrežo 3×3.

Slike smo nalagali in procesirali asinhrono preko korutin in s tem zagotovili odzivnost uporabniškega vmesnika.

Za dodatne pohitritve smo vse slike pretvorili v lasten format, ki nam omogoča uporabo hitrejših Unityjevih funkcij.

Ključne besede: Unity, DOF, Interaktivna, Vizualizacija, Ortofoto, Teren, Slovenija, LIDAR, Ločljivostni nivoji.

Abstract

Title: Interactive visualization of spatial data in Unity3D

Author: Nejc Smrkolj Koželj

In our thesis, we developed a prototype of an interactive 3D terrain visualization of Slovenia. We merged LiDAR data in form of height maps with orthophoto images, which were downloaded from the Slovenian Environment Agency's servers. We developed the prototype in Unity3D with programming language C#.

For successful interactive visualization, we implemented multiple support systems. We divided the terrain into tiles. Each tile consists of metadata and terrain. The terrain was generated from height maps and ortophoto images. For correct alignment, we implemented an offset alignment system and control point system.

Because of a high number of tiles, we implemented a level of detail system. The closer the camera is to the terrain, the higher resolution tiles are loaded and shown. The system also helped us with better memory management, as there are fewer tiles loaded in the memory. Tiles are linked together into a hierarchy, which is managed by a data structure called TreeStructure, which is grouping tiles into 3×3 grid.

Images were loaded and processed asynchronously, via coroutines, which helped us maintain a responsive user interface.

For additional performance gains, all images were converted to a proprietary format, which enabled us to use faster Unity functions.

Keywords: Unity, DOF, Interactive, Visualization, Orthophoto, Terrain, Slovenia, LIDAR, LOD.

Poglavje 1

Uvod

Vizualizacija nam je že od nekdaj služila kot učinkovit način podajanja tako abstraktnih kot konkretnih idej. Dandanes najdemo primere vizualizacije povsod okoli nas. S pomočjo vizualizacije lahko kompleksne in zapletene stvari prikažemo na človeku razumljiv način.

V večini primerov vizualizacije gre za lažje razumljiv prikaz informacij, kot je na primer vizualizacija zemljevida ali naravnega pojava. Vizualiziramo lahko tudi stvari, ki bi bile v naravnem svetu težko, če že ne nemogoče izvedljive. Primer tega je vizualizacija kompleksnega fizikalnega poskusa.

V dobi računalništva in modernih tehnologij se je možnost dobre vizualizacije precej približala človeku. Lažje je vizualizirati kompleksne pojave, poleg tega pa so takšne vizualizacije dostopne vsaki osebi. Z razvojem tehnologije so postale računalniške vizualizacije del našega vsakdanjika. Poleg tega je računalništvo vplivalo na razvoj interaktivne vizualizacije. Pri interaktivni vizualizaciji prikazana scena ni statična, ampak se spreminja glede na vnos uporabnika. Primer takšne vizualizacije so današnji spletni zemljevidi, po katerih lahko na primer iščemo in označujemo kraje ter merimo razdalje.

S pomočjo vizualizacije površja zemlje dobimo lažjo predstavo o planetu, na kateremu živimo. Pri vizualizaciji zemeljskega površja v tridimenzionalnem prikazu lahko dobljene informacije uporabimo za različne industrijske in raziskovalne namene.

1.1 Cilji diplomskega dela

1.1.1 Interaktivna vizualizacija terena Slovenije

Glavni cilj diplomskega dela je bil izdelati prototip interaktivne vizualizacije 3D terena Slovenije. Teren je bil sestavljen iz višinskih slik in ortofoto posnetkov, ki skupaj tvorijo 3D sceno. Interaktivno se uporabnik lahko skozi sceno premika ter si ogleduje različne dele terena.

Za omogočanje takšne interaktivne vizualizacije smo implementirali različne podporne sisteme. Implementirati smo morali sistem za ujemanje višinskih slik in ortofoto posnetkov.

Nalaganje terena je potekalo po ločljivostnih nivojih, pri čemer so deli terena, ki so od kamere dlje, prikazani v nižji resoluciji, deli terena, ki so kameri bližje, pa so prikazani v višji resoluciji. S tem smo poskušali optimizirati porabo pomnilnika in pospešiti 3D prikaz v pogonu.

Nalaganje slik smo poskušali izvesti paralelno, saj bi s tem lahko pohitrili čas nalaganja in obdelovanja slik ter izboljšali uporabniško izkušnjo.

Končna delujoča aplikacija se bo kasneje lahko integrirala skupaj z diplomsko nalogo 3D vizualizacije dogajanja na bojišču [11]. Prav tako bo služila kot osnova aplikacijam, ki bodo potrebovale interaktivno vizualizacijo 3D terena Slovenije.

1.1.2 Pilotska študija pogona Unity

Za implementacijo aplikacije smo želeli uporabiti pogon Unity. Skozi raziskavo smo želeli videti, kako se pogon Unity obnese pri interaktivni vizualizaciji večje količine dinamično naloženih terenskih objektov. Cilj je bil zadostiti čim bolj realno časovnemu prikazu, saj je aplikacija namenjena končnim uporabnikom. Za doseganje dobre uporabniške izkušnje sta potrebna odziven uporabniški vmesnik ter hitro nalaganje.

Zavedali smo se Unityjeve največje omejitve: izredno omejena uporaba paralelizma ter velike performančne težave [19]. Operacije kreiranja objektov

so performančno zelo potratne, paralelno jih pa ni mogoče izvajati zaradi omejitve samega pogona.

1.2 Pregled področja

Začetki vizualizacije terena so se začeli s kartami in zemljevidi. Zgodovina kartografije je bogata [4], razvijala se je sočasno in neodvisno na različnih koncih sveta. Najstarejši zemljevid do sedaj je bil najden v jami Lascaux v Franciji [20]. Zemljevid prikazuje položaje zvezd na nebu in je bil vklesan v skalo.

Dandanes si je težko predstavljati življenje brez zemljevidov. Uporabljamo jih za osnovno navigacijo po krajih, pri vožnji in za predstavo površja zemlje. Pri tem uporabljamo najrazličnejše zemljevide glede na namen uporabe. Nekateri prikazujejo le shemo terena, nekateri nam podajajo višinske podatke in nekateri so narejeni s pomočjo ortofoto posnetkov.

S časom so se razvili maketni zemljevidi, ki so navadne dvodimenzione zemljevide spremenili v tridimenzionalne. Iz njih je možno pridobiti informacijo o višini, kar še dodatno omogoča lažjo predstavo o terenu. 3D vizualizacije se v fizičnem svetu ne uporabljajo tako pogosto, saj so zelo nepraktične za transport in rokovanje. Možno jih je vizualizirati v digitalni obliki, kjer imajo veliko večjo uporabnost.

Na področju urbanističnega načrtovanja lahko 3D vizualizacija pomaga načrtovalcem. V raziskovalni nalogi *Integration of lidar and airborne imagery for realistic visualization of 3D urban environments* [9] je bila s pomočjo LiDAR višinskih podatkov, ortofoto slik in fotogrametrije narejena realistična vizualizacija mesta. Uporabnik se lahko skozi mesto premika v realnem času.

Z vizualizacijo naravnih nesreč lahko bolje razumemo vzroke za njihov nastanek. Poleg tega lahko ocenimo posledice ob morebitni ponovitvi. S pomočjo simulacijskih podatkov in zgodovinskih dejstev je bilo v *Visualizing the ground motions of the 1906 San Francisco earthquake* [6] natančno vizualizirano premikanje tal med potresom.

V astronomiji se interaktivna vizualizacija uporablja za 3D vizualizacijo vesolja. WorldWide Telescope omogoča prikaz nebesne panorame[1]. Slike neba so pridobljene s pomočjo zemeljskih in vesoljskih teleskopov.

S pomočjo interaktivne vizualizacije površja Marsa lahko NASA planira in optimizira vožnjo kopenskih vozil [17]. Teren generira s pomočjo satelitskih in kopenskih posnetkov površja. Pri tem je vizualizacija natančna do 0.01 metra.

Skupaj z drugimi raziskovalnimi inštitucijami ima NASA zbrane višinske podatke za celotno Zemljo [2] [2, 3]. Ti podatki so v veliki meri tudi javno dostopni. Uporabljeni so v aplikaciji Google Earth[3], kjer si uporabnik lahko interaktivno ogleduje površje Zemlje v 2D in 3D. Pri tem se realnočasovno nalagajo višinske slike in ortofoto posnetki.

Na področju Slovenije je bilo narejeno izredno malo interaktivne vizualizacije. V diplomski nalogi *Porazdeljeno sledenje žarkov za upodabljanje lasersko zajetih prostorskih podatkov* [13] je bil teren Slovenije vizualiziran kot svet računalniške igre *Minecraft*[4].

Naši nalogi najbližje nam je bila seminarska naloga *Vizualizacija terena v Unity3D* [14], iz katere smo tudi izhajali. V sklopu seminarske naloge so bili pretvorjeni LiDAR višinski podatki iz tekstovne datoteke v format PNG, pri čemer so barvni biti predstavljali višino terena v posameznih točkah.

[1] WorldWide Telescope spletna stran: http://www.worldwidetelescope.org/
[2] NASA višinske slike: https://visibleearth.nasa.gov/view.php?id=73934
[3] Google Earth spletna stran: https://www.google.com/earth/
[4] Minecraft spletna stran: https://minecraft.net/en-us/

Poglavje 2

Uporabljene tehnologije in orodja

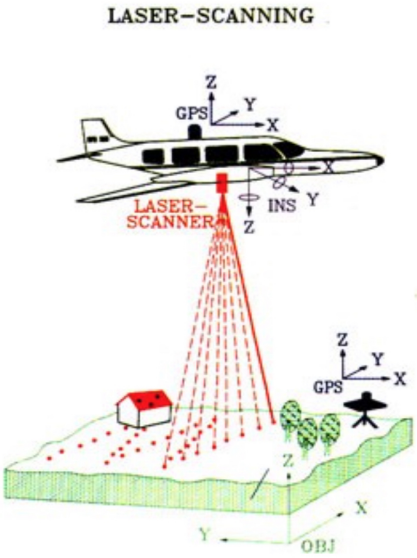
2.1 LiDAR podatki

LiDAR (angl. Light Detection and Ranging) [12] je način oziroma metoda za merjenje razdalje s pomočjo laserskih žarkov. Proti objektu se pošlje več laserskih žarkov, nato pa se spremlja njihov odboj. S pomočjo valovne dolžine žarka in razlike v času med poslanim in prejetim žarkom je možno izračunati razdaljo.

LiDAR je zelo uporabljena tehnologija predvsem v geografiji, geodeziji, arheologiji, geologiji in pri ostalih sorodnih vedah. Uporablja se tudi za vizualizacijo terena, za vožnjo avtonomnih vozil, optimizacijo vetrnih in sončnih elektrarn in v robotiki.

LiDAR sistem je sestavljen iz štirih sestavnih delov:

- Laser za ustvarjanje laserskih žarkov
- Skener in optika za obdelavo prejetih podatkov
- Fotodetektor in sprejemnik za sprejem žarkov
- GPS sistem za ugotavljanje položaja LiDAR naprave



Slika 2.1: Prikaz LiDAR skeniranja [5].

```
513047.79;30986.05;175.76
513047.79;30987.05;175.61
513047.79;30988.05;175.53
513047.79;30989.05;175.46
513047.79;30990.05;175.43
513047.79;30991.05;175.40
513047.79;30992.05;175.28
513047.79;30993.05;175.23
513047.79;30994.05;175.14
513047.79;30995.05;175.15
513047.79;30996.05;175.16
513047.79;30997.05;175.08
513047.79;30998.05;175.06
513047.79;30999.05;175.03
513048.80;30515.05;250.90
513048.80;30516.05;250.75
513048.80;30517.05;250.43
513048.80;30518.05;250.30
513048.80;30519.05;250.00
513048.80;30520.05;249.90
513048.80;30521.05;249.64
```

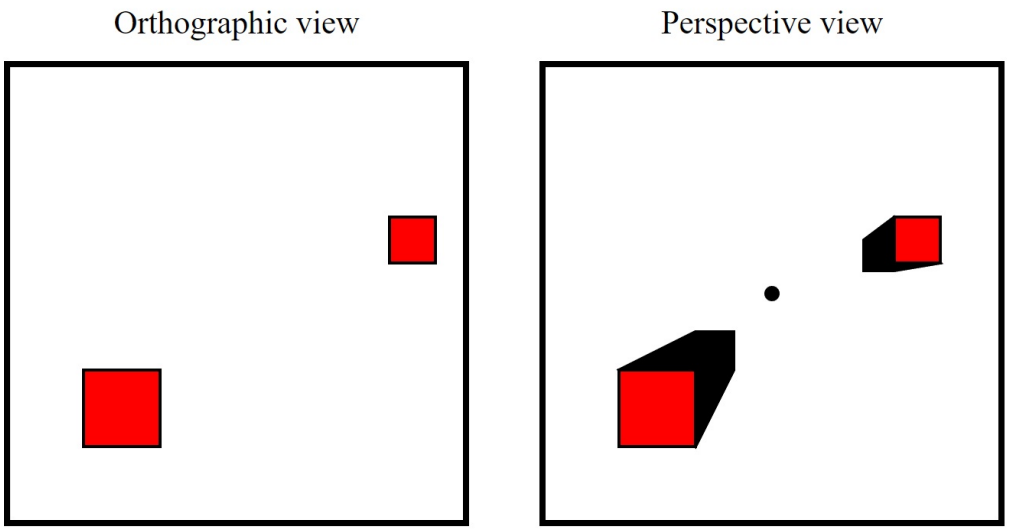
Slika 2.2: Primer LiDAR zapisa iz datoteke. Prva vrednost pove X koordinato GK, druga vrednost pove Y koordinato GK, tretja vrednost je višinski podatek.

2.2 Ortofoto posnetki

Ortofoto posnetki so zračne slike površja Zemlje, ki so obdelane s posebnimi geometričnimi postopki. Zajete so iz zraka, predvsem iz letala.

V geometrični obdelavi se perspektivno projekcijo spremeni na ortografsko, tako da so vsi objekti prikazani, kot da bi bili zajeti pod kotom 90°.

Zaradi popravkov na sliki predstavljajo natančno sliko zemeljskega površja in se uporabljajo za merjenje razdalj. Poleg tega jih lahko vizualiziramo v 3D okolju.



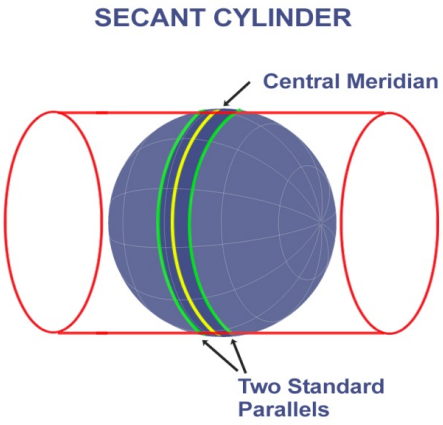
Slika 2.3: Prikaz primerjave ortografske in perspektivne projekcije [21].

2.3 Gauss-Krügerjev koordinatni sistem

Večinoma se vsakodnevno srečujemo s kartezičnim koordinatnim sistemom, pri čemer imamo dve ali tri osi, ki nam predstavljajo prostor. Takšna predstavitev je preprosta, težave pa nastanejo pri predstavitvi površine neravnih objektov. Zemlja je elipsoidne oblike, zaradi česar je nima smisla projectirati na ravnino, saj pri tem pride do napak. Zato se uporabljajo različni koor-

dinatni sistemi, ki bolje rešujejo problem ukrivljenosti in občutno zmanjšajo napake, ki nastanejo pri projiciranju.

V Sloveniji je zaradi zgodovinskih razlogov uveljavljen Gauss-Krügerjev koordinatni sistem (koordinatni sistem GK). Sistem je po delovanju podoben koordinatnemu sistemu, ki ga dobimo z uporabo univerzalne prečne Mercatorjeve projekcije (angl. Universal Transverse Mercator system) [8], le da je razlika med centralnimi medianami 3° v nasprotju z UTM, kjer je razlika 6°. Pri UTM se Zemljo razdeli v šestdeset con, ki so med seboj odmaknjene za 6°. Za vsako cono se površje Zemlje projicira na plašč valja s pomočjo sekajoče prečne Mercatorjeve projekcije.



Slika 2.4: Prikaz projiciranja ene cone UTM na valj [8].

2.4 Formata PNG in RAW

Slike so v računalništvu predstavljene na različne načine. Za našo nalogo je pomemben način, pri katerem je slika sestavljena iz pikslov. Posamezen piksel sestavljen iz treh kanalov, to je iz rdeče, zelene in modre barve.

PNG (angl. Portable Network Graphics) je razširjenih format zapisa slik. Format uporablja brezizgubno stiskanje. To pomeni, da ima končna slika enako kvaliteto kot pred pretvorbo, ampak potrebuje manj prostora [16]. To doseže z pomočjo filtriranja in algoritma DEFLATE.

Ortofoto posnetke, ki smo jih prejeli preko spleta, smo dobili v formatu PNG. Vnaprej izračunane višinske slike so bile prav tako shranjene v formatu PNG. Unityjevi objekti za upravljanje s teksturami kot osnovni format podpirajo PNG.

Zaradi Unityjevih performančnih težav pri nalaganju in uporabi formata PNG, smo slike pretvorili v naš poseben format, ki smo ga za potrebe naše aplikacije poimenovali RAW. Ta nima nobene povezave z že obstoječim standardnim formatom RAW. V tem formatu smo PNG pretvorili nazaj v osnovni zapis, pri čemer je bila datoteka sestavljena iz bitnega podatkovnega toka, v katerem so posamezni biti v seriji predstavljali podatke. Zaradi hitrejše obdelave v Unityju se nam je uporaba formata RAW kljub povečanemu času nalaganja vseeno izplačala.

Na primer, pri ortofoto posnetkih je vsak piksel sestavljen iz treh barv. V formatu RAW 8 bitov predstavlja posamezno barvo, poleg treh barv smo na koncu dodali še alfa kanal, ki pove transparentnost samega piksla. Skupaj 32 bitov smo v seriji zapisali v datoteko brez presledkov ali kakšnih drugačnih ločil. Velikost datoteke je zaradi tega občutno narasla in sicer iz okvirno 8 MB na 16 MB.

2.5 3D pogoni

Na področju vizualizacije obstaja že veliko orodij, ki so namenjena in prilagojena za določeno področje. Večnamenska orodja 3D vizualizacije zajemajo predvsem 3D pogone. Gre za aplikacije, s pomočjo katerih si lahko zelo pohitrimo in olajšamo razvoj prikaza 3D scen, objektov ter interakcij med njimi.

Zaradi njihovih prednosti in učinkovitosti se veliko uporabljajo v igričarski industriji. Razvoj računalniških iger se tako zelo pohitri, na projektih pa lahko brez težav sodelujejo tudi osebe brez programerskega oziroma matematičnega znanja. Na tržišču je več različnih pogonov, ki so posebej prilagojeni za različne žanre iger ter za različne platforme, na katerih igre tečejo.

Podjetja, ki razvijajo pogone, le te optimizirajo in prilagajajo za njihov ciljni žanr in platformo.

2.5.1 Unity3D

V naši nalogi smo uporabljali pogon Unity3D[1]. Pogon je v osnovi namenjen igričarski industriji in je v lasti podjetja Unity Technologies. Glavna prednost pogona Unity je možnost razvoja za različne operacijske sisteme in platforme, kot so PC, konzole, mobilne naprave ter spletne strani.

Unity omogoča programiranje v programskih jezikih C#, JavaScript ter Boo. Na voljo je pod različnimi licencami. Osnovna osebna licenca je na voljo brezplačno, ostale licence pa so plačljive, pri čemer je njihova glavna prednost boljša podpora, boljša orodja za performančno testiranje ter dostop do izvorne kode.

2.5.1.1 Korutine

Unity podpira paralelizem zelo omejeno, omogočeno je le paraleliziranje s pomočjo niti [1], kjer so podprte le najosnovnejše funkcionalnosti. Poleg tega se Unityjevi objekti in funkcije lahko uporabljajo le znotraj glavne niti.

Unity pa omogoča asinhrono izvajanje, katerega doseže z uporabo korutin (angl. Coroutine) [15]. Glavna ideja korutine je, da lahko izvajanje sredi korutine ustavimo ter ga kasneje nadaljujemo. Na ta način lahko ročno nadzorujemo odzivnost glavne niti, kar pomeni tudi odzivnost samega uporabniškega vmesnika.

Ko ustvarimo korutino, se ustvari kontekst, v katerem teče korutina. Ob ustavitvi korutine se kontekst korutine shrani in nadaljuje se izvajanje predhodnega konteksta. Nalaganje ter shranjevanje konteksta terja nekaj časa, ki pa pri velikemu številu korutin ni zanemarljiv [7].

V računalniških igrah, za katere je Unity tudi namenjen, je odzivnost glavne niti zelo pomembna. V igrah je pomembna hitrost posodabljanja

[1]Unity3D spletna stran: https://unity3d.com/

glavne niti, pri čemer po navadi ne potrebujemo več kot 60 posodobitev na sekundo oziroma 60 FPS (angl. Frames per Second). S pomočjo korutin se glavno nit lahko obdrži odzivno, medtem pa se lahko izkoristi ves čas, ki je še ostal po vseh posodobitvah.

Poglavje 3

Implementacija

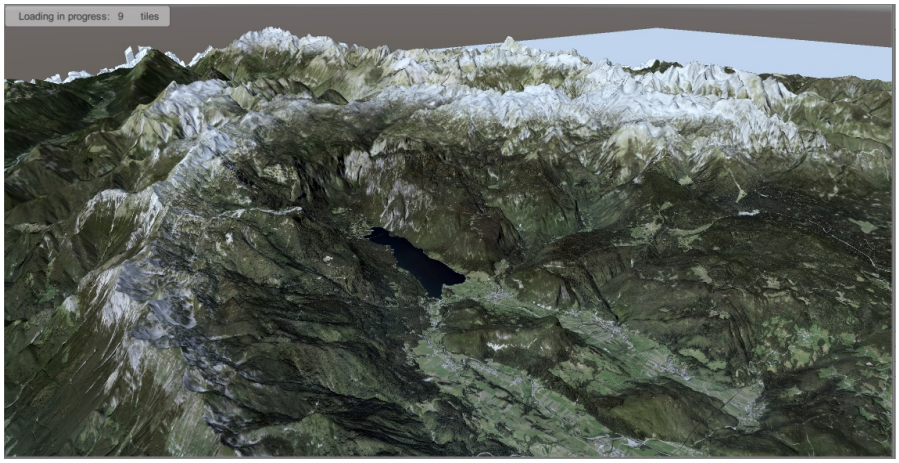
3.1 Pregled aplikacije

V aplikaciji omogočamo interaktiven prikaz 3D terena Slovenije. Teren Slovenije smo razdelili na več plošč. Ploščo sestavljajo metapodatki in teren, ki je generiran iz višinskih slik in ortofoto posnetkov, katere smo pridobili s strežnikov Agencije Republike Slovenije za okolje. Višinske slike in ortofoto posnetke smo poravnali. Za zagotavljanje tega smo implementirali sistem poravnave z odmikom in sistem kontrolnih točk. Pri tem smo se izognili kompleksni pretvorbi med koordinatnim sistemom GK in kartezičnim koordinatnim sistemom.

Zaradi velike količine plošče nalagamo v ločljivostnih nivojih. Podrobnejše plošče nalagamo glede na oddaljenost od kamere. Pri tem optimiziramo porabo pomnilnika. Za potrebe tega smo implementirali lastno podatkovno strukturo TreeStructure, ki združuje plošče v mrežo 3×3.

Plošče smo nalagali in procesirali asinhrono s pomočjo korutin. S tem smo obdržali uporabniški vmesnik odziven. Za hitrejše nalaganje in razbremenitev omrežnega prometa smo slike predpomnili lokalno in jih pretvorili v lasten format RAW.

Aplikacija omogoča tudi branje vseh nastavitev iz konfiguracijske INI datoteke (glej poglavje 6.1), kjer je mogoče nastaviti različne parametre.



Slika 3.1: Zaslonska slika aplikacije. V zgornjem levem kotu se vidi število plošč v nalaganju.

V okvirju v zgornjem levem kotu lahko uporabnik spremlja, koliko plošč trenutno nalagamo. V primeru napak v delovanju aplikacije ali ob njenem zagonu se pod tem okvirjem pojavi nov okvir z sporočilom napake. Uporabnik lahko obrača kamero s pritiskom na desni miškin gumb. S pomočjo vrtenja koleščka se lahko premika naprej in nazaj v smeri pogleda. Če pa miškin kolešček drži, se lahko pomika dvodimenzionalno glede na teren.

3.2 Nalaganje podatkov

3.2.1 Nalaganje plošč

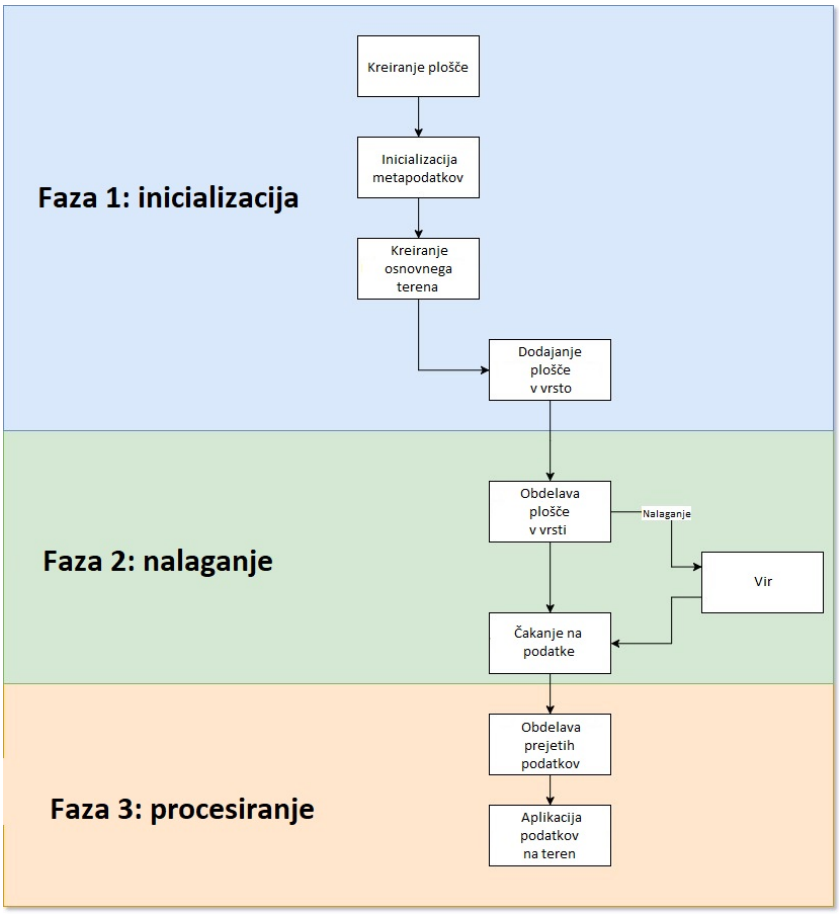
Plošče (angl. Tile) tvorijo osnovno strukturo za delovanje naše aplikacije. Sestavljene so iz terena ter metapodatkov.

Nalaganje plošče poteka v treh fazah. V prvi fazi naredimo ploščo ter inicializiramo metapodatke. V metapodatkih je zabeležen položaj plošče ter njeno vedenje v sceni. Shranijo se koordinate, velikost terena, ločljivost, ki jo vsebuje teren ter kateri ločljivostni nivo prikazuje. Poleg tega so shranjeni tudi podatki o stanju terena, ali je sploh naložen, kakšno je stanje nalaganja ter ali je teren prikazan ali skrit.

V koncu prve faze ploščo naložimo v posebno čakalno vrsto. Iz te vrste v glavnem programu jemljemo plošče, na katerih nato izvedemo nalaganje druge in posledično tretje faze. S pomočjo takega sistema lažje nadzorujemo, kdaj in kako naj se nalagajo plošče.

V drugi fazi naložimo višinske slike in ortofoto posnetke iz vira. Ortofoto posnetke nalagamo iz oddaljenega vira z ARSO-vih spletnih strežnikov. Višinske slike hranimo lokalno, v našem primeru na trdem disku.

V tretji fazi slike sprocesiramo ter apliciramo na teren. Glavni del procesiranja poteka preko Unityjevih funkcij za delo s terenom.



Slika 3.2: Diagram faz nalaganja plošče.

3.2.1.1 Pridobivanje podatkov

Vse potrebne podatke smo pridobili iz strežnikov Agencije Republike Slovenije za okolje (ARSO)[1]. Na njihovih strežnikih je možno najti razne podatke v povezavi s slovenskim okoljem[2]. Za nas so bili najbolj uporabni podatki LiDAR ter ortofoto posnetki Slovenije.

Podatke smo pridobivali preko spletne storitve REST (angl. Representational State Transfer) [18]. REST je zelo razširjen način pridobivanja podatkov oziroma komuniciranja preko interneta med različnimi napravami. S pomočjo URL se pove naslov naprave, željeno funkcionalnost in potrebne parametre.

Parametri, ki smo jih podali k zahtevi REST, so bili začetne in končne koordinate, željena ločljivost in format slike. Uporabljali smo največjo možno ločljivost in sicer 2048×2048. Za format slike pa smo želeli imeti PNG.

```
Whole REST query
gis.arso.gov.si/arcgis/rest/services/AO_DOF_2009_2011_AG101/MapServer/export?
bbox=
374372%2C
30513%2C
624118%2C
195517&bboxSR=&layers=&layerDefs=&size=
512%2C512
&imageSR=&format=jpg
&transparent=false&dpi=&time=&layerTimeOptions=&dynamicLayers=&gdbVersion=&mapScale=&f=image

Decomposed REST query
gis.arso.gov.si/arcgis/rest/services/AO_DOF_2009_2011_AG101/MapServer/export?
bbox=
374372 + %2C
30513 + %2C
624118 + %2C
195517 + &bboxSR=&layers=&layerDefs=&
size= + 512 + %2C + 512
&imageSR=&format= + jpg
&transparent=false&dpi=&time=&layerTimeOptions=&dynamicLayers=&gdbVersion=&mapScale=&f=image

Decomposed REST query with variables
gis.arso.gov.si/arcgis/rest/services/AO_DOF_2009_2011_AG101/MapServer/export?
BOX_COORDINATES_PARAMETER =
START_X + BOX_DATA_TERMINATOR
START_Y + BOX_DATA_TERMINATOR
END_X + BOX_DATA_TERMINATOR
END_Y + SOME_METADATA_1
BOX_SIZE_PARAMETER = TEXTURE_RES_X + BOX_DATA_TERMINATOR + TEXTURE_RES_Y
IMAGE_FORMAT_PARAMETER= + FORMAT
SOME METADATA 2
```

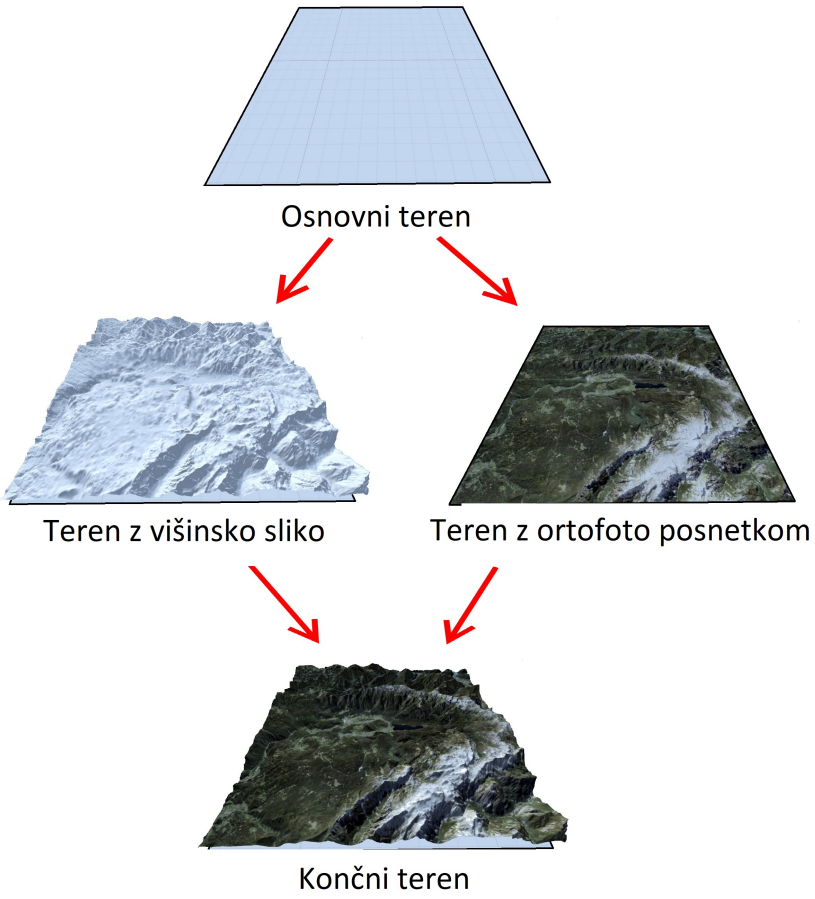
Slika 3.3: Prikaz REST klica iz naše aplikacije.

[1]ARSO spletna stran: http://www.arso.gov.si/

[2]ARSO strežniki: http://gis.arso.gov.si/arcgis/rest/services

3.2.1.2 Nalaganje in procesiranje slik

Teren je sestavljen iz višinskih slik in ortofoto posnetkov. Višinske slike so zgrajene na podlagi višinskih podatkov LiDAR, ortofoto slike pa na podlagi zračnih slik površja Slovenije.



Slika 3.4: Prikaz sestavljanja terena iz višinskih slik in ortofoto posnetkov.

Višinske slike beremo lokalno. Shranjene so v formatu PNG, katerega barvni biti povedo informacijo o višini v določeni točki. Ko se slika naloži, je potrebno barvne bite pretvoriti v višinski podatek, kjer zeleni barvni kanal predstavlja zgornjih 8 bitov 16 bitnega celega števila, modri barvni kanal pa spodnjih. Unityjeve funkcije zahtevajo razpon višinskih podatkov med 0 in 1, zato višinske podatke preslikamo na ta interval. Pri tem 1 predstavlja

največje možno 16 bitno število. Višinske podatke razporedimo v dvodimenzionalno matriko, katero apliciramo na teren s pomočjo Unityjevih funkcij.

Teksturne podatke pridobimo iz ortofoto posnetkov. Dostopni so na ARSO-vih strežnikih. Posnetke pridobivamo preko REST klicev v formatu PNG. Kot parametre pri REST klicu je potrebno podati ustrezne začetne in končne koordinate GK.

Med koordinatnim sistemom GK in kartezičnim koordinatnim sistemom smo naredili linearno pretvorbo, saj so napake glede na velikost Slovenije majhne. S pomočjo minimalnih in maksimalnih koordinat GK ter preko koordinat in velikosti plošče smo izračunali začetne in končne koordinate GK.

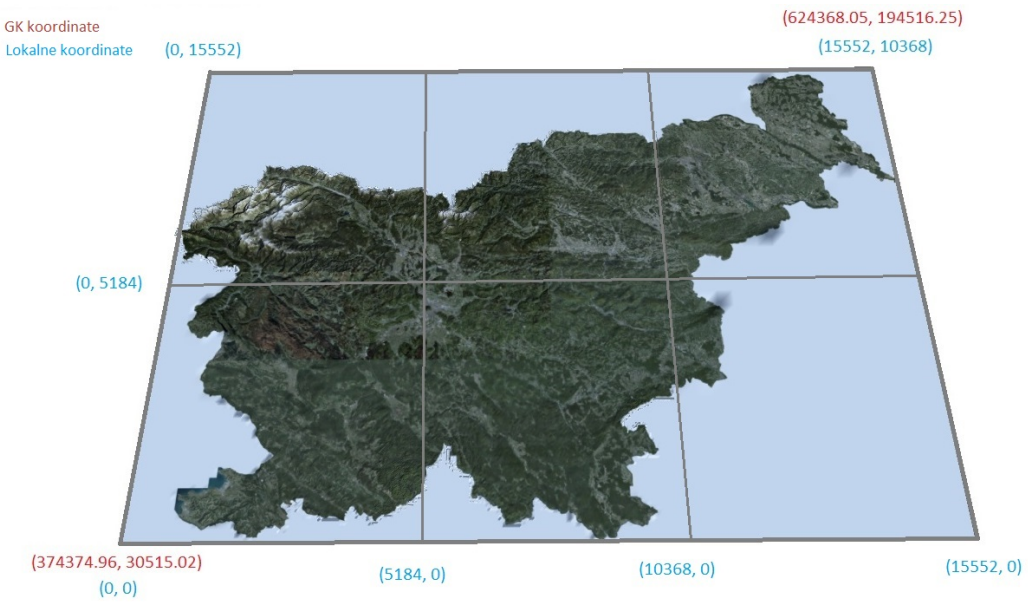
$$GKXStart = GKMinX + \frac{GKMaxX - GKMinX}{totalTerrainXSize} \times tileXCoordinate \tag{3.1}$$

$$GKZStart = GKMinZ + \frac{GKMaxZ - GKMinZ}{totalTerrainZSize} \times tileZCoordinate \tag{3.2}$$

$$GKXEnd = GKMinX + \frac{GKMaxX - GKMinX}{totalTerrainXSize} \times (tileXCoordinate + tileSize) \tag{3.3}$$

$$GKZEnd = GKMinZ + \frac{GKMaxZ - GKMinZ}{totalTerrainZSize} \times (tileZCoordinate + tileSize) \tag{3.4}$$

V odgovoru REST klica prejmemo ortofoto sliko. To s pomočjo Unityjevih funkcij nalepimo na teren.



Slika 3.5: Primer koordinat GK in lokalnih kartezičnih koordinat. Na sliki so višinske slike in ortofoto posnetki že poravnani.

3.2.1.3 Predpomnjenje slik

Zaradi počasnosti branja ter procesiranja formata PNG v pogonu Unity predpomnimo slike v našem lastnem formatu RAW.

Pri višinskih slikah barvni biti predstavljajo višinske podatke. Te podatke smo pretvorili v podatkovni tok, kjer so višinski podatki zapisani kot serija bitov. Po 16 bitov tvori en podatek o višini. Branje takšne datoteke je bistveno hitrejše, saj so Unityjeve funkcije pri branju in procesiranju PNG slik počasne.

Ortofoto slike smo ob naložitvi iz oddaljenega vira predpomnili v lokalni vir. S tem smo razbremenili omrežni promet, ki bi nastal ob vsakokratnem nalaganju istih ortofoto slik iz oddaljenega vira. Poleg tega smo slike sočasno tudi pretvorili v naš format RAW. Podobno kot pri višinskih slikah smo tudi tu imeli tok podatkov, le da smo za posamezni piksel slike porabili 32 bitov. Po 8 bitov zavzemajo trije barvni kanali. Četrti, alfa kanal, predstavlja v

Unityju odboj svetlobe. Tega nismo želeli imeti, zato smo kanal nastavili na 0.

Pridobitev pri predpomnjenju ortofoto posnetkov je tudi v tem, da ni več potrebno računati pripadajočih koordinat GK. Predpomnjene ortofoto slike se povezujejo z višinskimi slikami, tako da ima vsaka višinska slika svojo ortofoto sliko. To povezavo ohranjujemo z podobnim poimenovanjem datotek na lokalnem viru.

3.3 Preračunane višinske slike

Višinske slike so zgrajene na osnovi LiDAR podatkov. Ti so zajeti v tekstovni ASC datoteki v Gauss-Krügerjevem koordinatnem sistemu. Format vsebuje sledeč zapis in je ločen le s podpičji: x koordinata; y koordinata; višina. Podatki so ločeni z novo vrstico. V naši nalogi smo izhajali iz že zgeneriranih višinskih slik ter ločljivostnih nivojev, kateri so bili narejeni v sklopu seminarske naloge *Vizualizacija terena v Unity3D* [14].

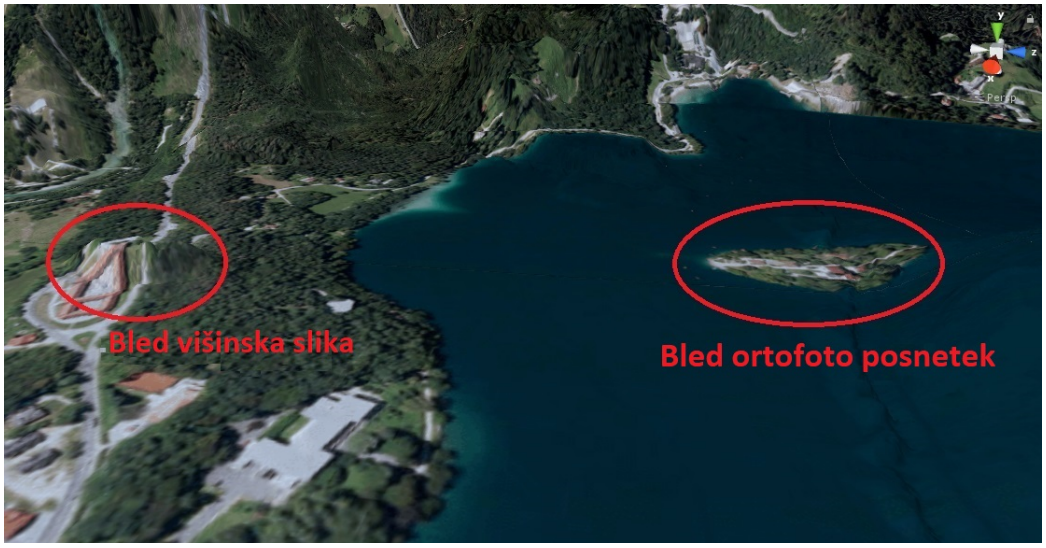
V sklopu te naloge so bili podatki pretvorjeni v format PNG, kjer sta dva barvna kanala po 8 bitov predstavljala višinski podatek. Posamezni pixel je predstavljal točko terena. Višinske slike so v ločljivosti 1024×1024 pikslov. Pri generiranju slik je prišlo do zaokrožitvene napake, kar se pozna na točnosti višinskih podatkov. Pri pretvorbi iz koordinatnega sistema GK v kartezični koordinatni sistem so bile koordinate zaokrožene iz decimalnih na celoštevilčne vrednosti, kar je povzročilo nenatančnost pri poravnavi z ortofoto posnetki za približno 1 meter.

Pri pretvorbi v format PNG so se pojavila tudi zelo majhna območja, za katera nismo imeli višinskih podatkov. Ta območja so bila popravljena z metodo najbližjih sosedov. Popravljene slike so bile nato po 3×3 združene v večje, a manj podrobne slike. Iz tega je sledilo 5 ločljivostnih nivojev.

3.4 Poravnava slik

Ortofoto posnetki so shranjeni v koordinatnem sistemu GK. Že zgenerirane višinske slike so shranjene v lokalnem kartezičnem koordinatnem sistemu.

Ko smo poskušali nalepiti ortofoto posnetke na teren, je prišlo do odstopanj pri poravnavi med višinskimi slikami in ortofoto posnetki. Vzrok za to so popravki, narejeni na višinskih slikah pri pretvorbi med koordinatnim sistemom GK in kartezičnim koordinatnim sistemom ter njihov konstanten zamik.



Slika 3.6: Primer neujemanja višinskih slik in ortofoto slik - Blejski otok.

3.4.1 Poravnava z odmikom

Najprej smo implementirali metodo, ki smo jo poimenovali Poravnava z odmikom. Uporabljene višinske slike smo zamaknili od začetnih koordinat GK za nek konstanten odmik.

Odmik smo določili ročno. Kot referenčno točko smo si izbrali Blejski otok, saj je poravnava višinskih ter ortofoto posnetkov tam najlažja. Jezero je ravno, otok pa izstopa. Ročno smo nastavili odmike, tako da so se višinske

slike ter ortofoto posnetki ujeli do metra natančno.

3.4.2 Kontrolne točke

Težava metode z odmikom je v tem, da napaka narašča z odmikom od referenčne točke. Na področju Slovenije je napaka majhna. V primeru, da je referenčna točka v središču Slovenije, bi prišlo na robovih Slovenije do največ nekaj metrov odstopanja. Napaka pa bi postala opazna ob razširitvi naše aplikacije na vizualizacijo terena sosednjih držav.

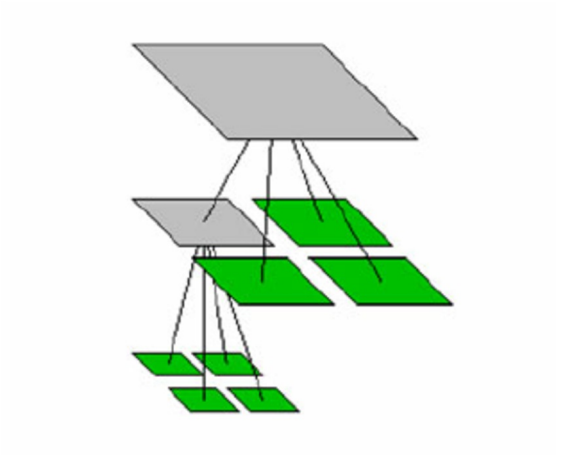
Kot nadgradnjo poravnave z odmikom smo implementirali kontrolne točke. Osnovna ideja o odmiku je tukaj enaka, le da imamo več vnaprej ročno določenih točk. Vsaka točka hrani koordinato GK ter njeno lokalno kartezično koordinato. V primeru, da imamo le eno kontrolno točko, bo sistem kontrolnih točk deloval enako kot pozicioniranje z odmikom.

Ko nalagamo ploščo, bodo koordinate GK poračunane za konstanten odmik od najbližje kontrolne točke. Kontrolne točke lahko ročno določimo le enkrat za različne dele Slovenije, kar zelo zmanjša napake pri pretvorbi iz enega sistema v drugega.

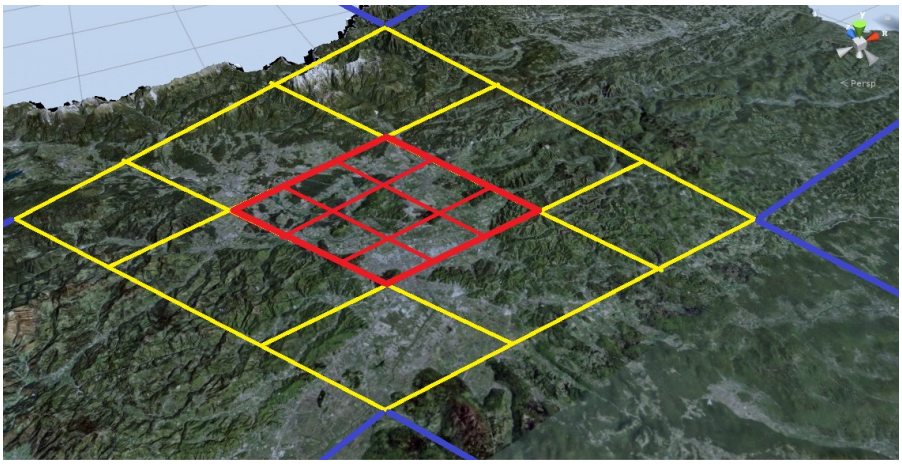
3.5 Predstavitev podatkov

Osnovni teren Slovenije sestavlja 39690 plošč. Pri pogledu na celotni teren Slovenije zaradi razdalje do kamere ni smiselno prikazati vseh plošč z najvišjo natančnostjo. V primeru, da je kamera daleč od plošč, združimo več sosednjih plošč v eno manj podrobno. Tako se plošče povezujejo v našo hierarhično podatkovno strukturo, poimenovano TreeStructure. V naši aplikaciji smo uporabili 5 ločljivostnih nivojev. Tako imamo na najvišjem nivoju le 6 plošč.

TreeStructure temelji na podatkovni strukturi QuadTree, katero pa smo za naše potrebe prilagodili. Glavna razlika je v tem, da se pri QuadTree elementi delijo v 2×2 mrežo, pri naši TreeStructure strukturi pa je ta mreža 3×3.



Slika 3.7: Prikaz hierarhije 2×2 pri podatkovni strukturi QuadTree [10].



Slika 3.8: Prikaz ločljivostnih nivojev ter njihova medsebojna 3×3 hierarhija.

Pri vizualizaciji terena je 3×3 mreža boljša kot QuadTree 2×2 mreža. Uporabnik lahko vidi sredinsko ploščo, pri rotaciji kamere pa lahko vidi tudi ostale plošče, ki mejijo na sredinsko ploščo. Pri premiku kamere za največ eno ploščo je premik neopazen, saj so sosednje plošče že naložene v enaki ločljivosti, kot je nivo pod kamero.

Vsak element strukture ima več podrejenih elementov strukture, ki vsebujejo podrobnejše plošče. Glede na približanje kamere posamezni plošči se le ta skrije, na njeno mesto pa se naloži 9 manjših, podrobnejših podrejenih

plošč, razporejenih v mrežo 3×3. Naložili smo nižji nivo. V nasprotnem primeru, ob oddaljitvi, se 9 podrejenih plošč skrije, na njihovem mestu pa se prikaže manj podrobna večja plošča.

V trenutku, ko je potrebno naložiti podrobnejši nivo, se začnejo nalagati podrejene plošče. Ob nalaganju ostanejo skrite. Pokažejo se šele, ko se jih naloži vseh 9. Takrat se nadrejena plošča skrije. To je potrebno zaradi različne razgibanosti terena pri različnih ločljivostnih nivojih, saj je lahko višina določene točke terena na višjem nivoju višja oziroma nižja kot pri nižjem nivoju.

3.6 Upravljanje s pomnilnikom

Količina porabljenega pomnilnika je odvisna od terena posamezne plošče. Teren plošče je odvisen od ločljivosti ortofoto posnetkov ter od ločljivosti višinskih slik. Za ortofoto posnetke ločljivosti 1024×1024 je poraba pomnilnika na ploščo 20 MB, pri ločljivosti 2048×2048 pa 35 MB. Zaradi pomnilniških omejitev nalaganje celotnega terena v pomnilnik ni možna saj bi potrebovali okvirno 1390 GB pomnilnika.

Pri porabi pomnilnika smo poskušali biti varčni. Strinjali smo se, da aplikacija ne sme porabiti več kot 8 GB. To omejitev smo izbrali na podlagi trenutnih pomnilniških zmožnosti namiznih računalnikov ter na predpostavki, da bo kasnejša morebitna integracija naše aplikacije le še pridobila na porabi.

Najnižjo porabo pomnilnika dosežemo, če pobrišemo iz pomnilnika vse plošče, ki se trenutno ne prikazujejo. Problem takšnega pristopa je, da imamo veliko število nalaganj, ki pa so časovno potratna. Zato plošč na višjih nivojih nismo brisali in s tem izboljšali uporabniško izkušnjo.

V aplikacijo smo vgradili možnost, da se že naložene plošče ne brišejo. Izhajali smo iz predpostavke, da bo v določenih primerih in okoliščinah uporabnik pregledoval le določeno geografsko področje in bo potrebno ponovno prikazati že naloženo ploščo.

S podobnim izhodiščem smo implementirali predpomnjenje plošč do določenega

nivoja. Pri tem se bodo naložile vse plošče do specificiranega nivoja, kar z vsakim nivojem eksponentno poveča porabo pomnilnika. To omogoča uporabniku prijaznejšo uporabo, predvsem na višjih nivojih.

Pri optimizaciji pomnilnika ni najboljše možne rešitve. V nekaterih primerih je smiselno obdržati porabo na minimumu, medtem ko v drugih primerih ni smiselno brisati posameznih plošč. V naši aplikaciji ponujamo oba sistema, uporabnik pa izbere sebi primernega.

3.7 Paralelno in asinhrono nalaganje

Glavni cilj asinhronega nalaganja je bil obdržati prosto glavno programsko nit za odzivnost uporabniškega vmesnika.

Največja omejitev sekvenčnega nalaganja je neodzivnost aplikacije. V času nalaganja plošče je aplikacija povsem neodzivna, kar pri nalaganju večjega števila plošč zelo kvari uporabniško izkušnjo. Poleg izboljšanja uporabniške izkušnje bi lahko z paralelnim nalaganjem izkoristili moderne večjedrne sisteme, saj bi lahko zelo pohitrili nalaganje.

Unityjeva podpora paralelnega programiranja je zelo omejena. Na voljo so najosnovnejše knjižnice za nitenje, vendar pa Unityjevi osnovni objekti in funkcije ne dopuščajo paralelizma z nitmi. Posledično to pomeni, da si z nitmi nismo mogli pomagati. Edino, kar smo imeli na voljo, so Unityjeve korutine, tako da smo se osredotočili predvsem nanje.

Ko se sproži nalaganje plošče, se nalaganje višinskih ter ortofoto posnetkov izvaja v korutinah. Najprej smo nalagali plošče sekvenčno, slike pa paralelno preko korutin. Ko se je ena plošča v celoti naložila, smo začeli nalagati naslednjo. Podobno smo implementirali sistem, kjer smo plošče nalagali paralelno, same slike za posamezno ploščo pa sekvenčno. Oba sistema sta prinesla zelo podobne rezultate, vendar sta bila od povsem sekvenčnega nalaganja počasnejša za približno 25%.

V naslednjem koraku smo oba sistema združili, pri čemer smo plošče in slike nalagali hkrati. Pri tem smo imeli največ uspeha s krajšim časom nala-

ganja, vendar na račun uporabniške izkušnje. V povprečju je čas nalaganja do 20% hitrejši od sekvenčnega, vendar pa je uporabniški vmesnik postal med časom nalaganja skoraj povsem neodziven.

Za doseganje odzivnega uporabniškega vmesnika smo implementirali sistem nalaganja s pomočjo vrste, ki je tekel v korutini. V njem vzamemo plošče iz vrste ter sploţimo nalaganje druge in tretje faze. Z dvema korakoma v globino smo obdržali uporabniški vmesnik odziven.

3.8 Parametrične možnosti

V razvoju aplikacije smo veliko razmišljali o kasnejši integraciji z drugimi projekti. Preko konfiguracijske INI datoteke (glej poglavje 6.1) je možno nastavljati različne konstante in parametre. Datoteka se prebere ob zagonu aplikacije.

Za premikanje kamere je možno določiti hitrosti premikanja, obračanja in približevanja. Poleg tega je možno invertirati smer premikanja.

Za plošče je možno nastavljati privzeto velikost, željeno ločljivost ortofoto posnetkov ter ločljivost višinskih slik. Poleg tega je možno izklopiti nalaganje višinskih slik ter ortofoto posnetkov. Za ortofoto posnetke je možno vklopiti predpomnjenje in pretvorbo v format RAW. Pri lokalnem predpomnjenju slik lahko spreminjamo pot do vira.

Pri poravnavi plošč lahko nastavljamo koordinatni obseg GK, pri čemer lahko določimo začetne in končne koordinate. Preklapljamo lahko med poravnavo plošč z odmikom in kontrolnimi točkami. Pri poravnavi plošč z odmikom lahko ročno nastavljamo referenčno točko.

Nalaganje plošč poteka sekvenčno, možno pa je vklopiti paralelno nalaganje. Pri tem se bodo plošče nalagale hitreje na račun uporabniške izkušnje. Možno je tudi izklopiti brisanje plošč, ki so že bile naložene.

Število ločljivostnih nivojev je možno nastavljati. Poleg tega je možno vklopiti prednaložitev vseh plošč do določenega nivoja.

3.9 Težave

3.9.1 Poravnava slik

Precej težav smo imeli pri poravnavi višinskih slik z ortofoto posnetki. Višinske slike so bile vnaprej izračunane v okviru druge raziskovalne naloge, tako da nismo točno vedeli začetnih in končnih koordinat GK. Posledično smo imeli težave pri ujemanju slik. Pri računanju višinskih slik iz LiDAR podatkov je bilo narejeno veliko zaokroževanja, kar je povzročilo majhne zamike slik glede na posamezno ploščo.

Poleg tega so vse višinske slike vsebovale nek začetni zamik. Začetna točka v višinskih slikah ni bila enaka začetni točki v ortofoto posnetkih. Zaradi tega smo implementirali pozicioniranje z odmikom, kar je v začetku rešilo ta problem. Sistem smo kasneje nadgradili v sistem kontrolnih točk, ki je natančnejši, saj omogoča določanje večih referenčnih točk na različnih delih Slovenije.

3.9.2 Unity

3.9.2.1 Počasne funkcije

Osnovne funkcije v Unityju so počasne. V drugi fazi se je opazila velika razlika pri branju z lokalnega vira, pri kateri smo uporabili standarndne C# funkcije in Unity funkcije. Na konkretnemu primeru nalaganja slik iz lokalnega vira smo testirali čas branja z diska. Našo RAW datoteko smo prebrali v 10-15 milisekundah, medtem ko je Unityjeva funkcija potrebovala kar 110-120 milisekund za branje slike PNG. Pri tem je potrebno dodati, da je RAW datoteka v povprečju dvakrat večja kot slika PNG.

Poleg tega je delovanje Unityjevih funkcij zelo odvisno od podanih parametrov. Na primer: inicializacija prazne teksture lahko porabi glede na parametre med 3 in 170 milisekundami. Veliko časa smo namenili ugotavljanju, katere Unityjeve funkcije bi lahko uporabili ter s kakšnimi parametri.

3.9.2.2 Asinhrono nalaganje

Paralelizem v Unityju je zelo slabo podprt. Na voljo so niti, vendar Unityjevi razredi ne dopuščajo izvajanja izven glavne niti. Posledično je za nas to pomenilo, da paraleliziranja z nitmi ne bomo mogli uporabljati. Večina naše aplikacije se nanaša na kreiranje in prikazovanje terena, kar pa skoraj v celoti uporablja Unityjeve objekte in funkcije. Poizkusili smo zaobiti te omejitve, toda pri tem nismo bili uspešni.

Sama uporaba korutin je tudi vplivala na performance aplikacije. Kreiranje le teh in preklapljanje med različnimi korutinami v ozadju je časovno potratno, tako da so pohitritve z korutinami majhne. Problem korutin je, da je potrebno narediti premor ročno. Do težav pride pri Unityjevih funkcijah. Skoraj vse funkcije za delo s terenom potrebujejo precej časa, pri čemer jih ni možno sredi izvajanja prekiniti. To se najbolj pozna pri slabi odzivnosti uporabniškega vmesnika ob nalaganju.

Nalaganje preko korutin tudi ni bilo dovolj za obdržanje odzivnosti aplikacije. Tukaj smo šli za dva koraka v globino in sicer smo imeli korutino, ki skrbi za nalaganje in hkrati tudi kliče posamezne korutine plošč. Prvotno smo implementirali sistem, ki bi skrbel za nalaganje plošč ter optimizacijo korutin, vendar pa smo ugotovili, da je sistem z dvema korakoma v globino učinkovitejši.

3.9.2.3 Puščanje pomnilnika

V tretji fazi procesiramo slike. Unity vsebuje hitre funkcije za obdelavo slik in aplikacijo slik na teren, vendar le te vsebujejo puščanje pomnilnika (angl. Memory leak). Do tega pride zaradi referenc kazalcev na same teksturne slike, pri čemer smetar (angl. Garbage Collection) ne počisti vseh tekstur in le te ostanejo v pomnilniku. Ker je procesiranje slik osnovni del naše aplikacije, ki se izvrši velikokrat, takšnih hitrih dostopov nismo mogli uporabljati. Namesto njih pa smo bili primorani uporabiti počasnejše Unityjeve funkcije.

Poglavje 4

Rezultati

Naš glavni cilj je bila interaktivna vizualizacija terena Slovenije s pogonom Unity. Za zagotavljanje tega je skrbelo več različnih skupaj povezanih sistemov.

Testiranje smo izvedli na stacionarnem računalniku s procesorjem Intel Core i7-6700K, Skylake, 4 GHz, 8 jeder (16 skupaj z Hyper Threadingom). Imeli smo 16 GB, DDR4, 2666 MHz pomnilnika. Uporabili smo grafično kartico AMD Radeon R9 390x z 8192 MB grafičnega pomnilnika. Pri nalaganju iz lokalnega vira smo uporabili HDD z 7200 RPM. Pri nalaganju iz oddaljenega spletnega vira smo uporabili internetno povezavo 20/2 MBit.

Teren Slovenije smo razdelili v več plošč. Ploščo sestavljajo metapodatki in teren, kateri je generiran iz višinskih slik in ortofoto posnetkov. Za poravnavo višinskih slik in ortofoto posnetkov smo morali implementirati sistem poravnavanja z odmikom. S pomočjo referenčne točke smo ujeli slike in se izognili kompleksni pretvorbi med koordinatnim sistemom GK, v katerem so bili ortofoto posnetki ter lokalnim kartezičnim sistemom, ki ga uporabljata Unity in preračunane višinske slike. V primeru, da je referenčna točka v središču Slovenije, bo prišlo do odstopanja na robovih Slovenije za par metrov. Za zmanjšanje te napake smo implementirali sistem kontrolnih točk. Ta deluje podobno kot sistem za poravnavanje z odmikom, le da imamo tukaj več referenčnih točk.

Višinske slike so v ločljivosti 1024×1024. Ločljivost ortofoto posnetkov lahko spreminjamo, največja možna ločljivost je 2048×2048. Poraba pomnilnika na ploščo je odvisna od ločljivosti ortofoto posnetkov. Pri ločljivosti 1024×1024 je poraba 20 MB, pri ločljivosti 2048×2048 pa 35 MB. Čas nalaganja in procesiranja ene plošče je približno 150 milisekund.

Ortofoto posnetke nalagamo iz ARSO-vih strežnikov. Naložene slike shranimo lokalno v formatu PNG, saj je čas nalaganja iz lokalnega vira občutno hitrejši, kot iz oddaljenega. Posledično razbremenimo omrežni promet. Poleg tega lahko pri branju iz lokalnega vira uporabljamo hitrejše funkcije za nalaganje. Pri shranjevanju ortofoto posnetek pretvorimo v naš format RAW, saj je procesiranje hitrejše zaradi hitrejših Unityjevih funkcij. Velikost slik v formatu PNG pri ločljivosti 2048×2048 je v povprečju okoli 8 MB, velikost slik v formatu RAW pa 16 MB. Čas nalaganja in procesiranja iz spletnega vira traja od 1 do 3 sekunde. Nalaganje in procesiranje formata PNG iz lokalnega vira traja med 50 in 150 milisekundami, format RAW pa potrebuje le 35 milisekund.

V naši raziskovalni nalogi smo uporabili višinske slike, ki so bile zgenerirane v okviru seminarske naloge *Vizualizacija terena v Unity3D* [14]. Slike smo dobili v formatu PNG, ki smo ga, tako kot pri ortofoto posnetkih, pretvorili v format RAW. Velikost slik v formatu PNG je med 1,1 in 1,5 MB, v formatu RAW pa 2 MB. Čas nalaganja formata RAW traja v povprečju 15 milisekund, čas procesiranja pa približno 85 milisekund.

Osnovni teren Slovenije sestavlja 39690 plošč. Za prikaz vseh plošč hkrati bi potrebovali približno 1390 GB pomnilnika. Poleg tega bi nalaganje in procesiranje slik trajalo 100 minut. Zaradi tega povezujemo plošče v ločljivostne nivoje, saj ni smiselno prikazati toliko plošč z najvišjo natančnostjo, ko je kamera daleč.

Za potrebe prikaza različnih ločljivostnih nivojev smo implementirali podatkovno strukturo TreeStructure. Vsaka plošča ima 9 podrejenih in bolj podrobnih plošč na nižjem nivoju. Bližje kot bo kamera, bolj podrobne plošče bomo prikazali. Velikost terena plošče na najnižjem nivoju je 64×64. Na

vsakem višjem ločljivostnem nivoju velikost plošče naraste za faktor 3. Naša aplikacija ima 5 ločljivostnih nivojev, tako da je velikost plošče na najvišjem nivoju enaka 5184×5184.

5 ločljivostnih nivojev smo si izbrali, ker je to največje možno število nivojev, če združujemo plošče v mrežo 3×3. Tako dobimo na najvišjem nivoju 6 plošč. Več kot je ločljivostnih nivojev, več prostora potrebujemo za shranjevanje slik. Poleg tega se plošče nalagajo bolj pogosto. Vendar pa je poraba pomnilnika v aplikaciji manjša, saj imamo hkrati naloženih manj plošč.

Več kot 5 nivojev na področju Slovenije ni smiselno uporabljati, saj bi morali slike združevati v mrežo 2×2. Pri tem se potreba po prostoru za shranjevanje slik zelo poveča, poveča pa se tudi število nalaganj. V zameno prihranimo malo pomnilnika. Imeti manj kot 4 nivoje ni smiselno, saj je pri 3. nivojih prikazanih na najvišjem nivoju 486 plošč. Za prikaz vseh bi potrebovali 17 GB pomnilnika, kar je veliko za prikaz le najvišjega nivoja. Pri 4. nivojih je potrebnih za prikaz najvišjega nivoja 54 plošč, za kar potrebujemo 1,9 GB pomnilnika. V naši aplikaciji smo uporabili 5 nivojev, saj je poraba pomnilnika pri tem najmanjša. Vgradili pa smo sistem za prednaložitev vseh plošč do določenega nivoja, saj je v primerih, ko imamo pomnilnika dovolj, smiselno imeti le 4 nivoje.

Plošče smo poskušali nalagati paralelno preko niti, vendar zaradi omejitev Unityja nismo bili uspešni. Nalaganje smo izvedli asinhrono s pomočjo korutin. Da smo uspeli obdržati uporabniški vmesnik odziven, smo implementirali sistem nalaganja z dvema korakoma v globino. Glavni program v korutini upravlja nalaganje korutin posameznih plošč. Plošče lahko nalagamo sekvenčno ali pa paralelno. V primeru paralelnega nalaganja je hitrost nalaganja do 20% hitrejša od sekvenčnega, vendar na račun slabše odzivnosti uporabniškega vmesnika.

Poglavje 5

Zaključek

Interaktivno smo vizualizirali 3D teren Slovenije z uporabo pogona Unity3D. Teren smo razdelili na plošče. Sestavili smo jih iz višinskih slik, ki so temelile na LiDAR podatkih, in ortofoto posnetkov. Za hitrejše branje in obdelavo smo slike predpomnili lokalno in jih pretvorili v naš format RAW.

Slike smo poravnali s pomočjo sistema poravnave z odmikom in sistema kontrolnih točk. Pri tem smo se izognili težavni pretvorbi med različnimi koordinatnimi sistemi z zanemarljivimi napakami.

Plošče smo nalagali v ločljivostnih nivojih, ki so se nalagali v odvisnosti od razdalje do kamere. S tem smo bolje izrabljali pomnilnik. Implementirali smo svojo podatkovno strukturo TreeStructure in sistem hierarhičnega nalaganja.

Z uporabo korutin smo nalagali in obdelovali slike paralelno. Tako smo obdržali uporabniški vmesnih odziven.

5.1 Nadaljnje delo

V naši nalogi smo uporabili višinske slike, ki so bile zgenerirane v okviru seminarske naloge *Vizualizacija terena v Unity3D* [14]. Pri generiranju je bilo narejenih veliko zaokroževanj pri pretvorbi med koordinatnimi sistemi in pri pretvorbi med različnimi podatkovnimi tipi. Zaradi tega je prišlo do odstopanj pri poravnavi in zamiku višinskih slik. Potrebno bi bilo ponovno

zgenerirati višinske slike.

Poravnavo je možno narediti bolj natančno z dodatnimi kontrolnimi točkami za različne dele Slovenije. S pomočjo tega se lahko še bolj zmanjša napaka pretvorbe med koordinatnimi sistemi. Poleg tega bi bilo potrebno zasnovati sistem, kjer bi lahko točke določiti matematično in ne ročno. Trenutno smo točke določili ročno, saj zaradi napak pri zaokroževanju natančnejša matematična aproksimacija ni bila smiselna.

Za boljšo uporabniško izkušnjo je potrebno razširiti sistem prednalaganja in mu dodati sistem za prednaložitev določenega področja. S tem lahko v specifičnih uporabah zelo izboljšamo uporabniško izkušnjo. Ta sistem je uporaben v aplikacijah, kjer je potrebno prikazati le določen del terena, prikaz podrobnejših plošč pa mora biti hiter. Za delovanje takšnega sistema potrebujemo velik pomnilnik, pri zagonu aplikacije pa določen čas, da se lahko kreirajo vse potrebne plošče in naložijo vse slike.

Največja performančna omejitev je, da Unity komajda omogoča paralelizem. Zato smo na račun daljšega časa nalaganja izboljšali odzivnost uporabniškega vmesnika. Prav tako je zaradi delovanja korutin nalaganje skoraj povsem sekvenčno, kar posledično pomeni, da je aplikacija veliko počasnejša, kot bi lahko bila. Problem nalaganja plošč je možno zelo enostavno paralelizirati, vendar pa bo potrebno počakati, da začne Unity pogon podpirati paralelizem.

Preučiti bi bilo potrebno nalaganje plošč v obliki samostojnih scen. V primeru, da je mogoče nalagati samostojne scene paralelno, je to možna rešitev omejenega paralelizma.

Unityjevi objekti in funkcije za obdelavo slik in delo s terenom so veliko počasnejše, kot bi lahko bile. Poglobiti se je treba v pogon ter zamenjati objekte in funkcije za nalaganje in obdelavo slik ter za obdelavo in prikaz terena. To bi lahko pohitrilo procesiranje za faktor 3. Poleg tega bi to lahko omogočilo uporabo paralelizma.

Poglavje 6

Dodatki

6.1 INI datoteka

Naši aplikaciji smo omogočili, da prebere vse konfiguracijske podatke iz INI datoteke. V njej je možno definirati glavne konstantne parametre.

Spremenljivka	**Tip**	**Razlaga**
LoadConfigurationFromFile	bool	Pove ali se naloži konfiguracija iz datoteke ali naj se naloži prednastavljena konfiguracija
MinimumCameraheight	int	Najnižja točka kamere
MaximumCameraheight	int	Najvišja točka kamere
CameraRotateXSpeed	float	Hitrost vrtenja kamere po X osi
CameraRotateYSpeed	float	Hitrost vrtenja kamere po Y osi
InvertCameraRotateX	bool	Obratno vrtenje kamere po X osi
InvertCameraRotateY	bool	Obratno vrtenje kamere po Y osi

CameraMoveXSpeed	float	Hitrost premikanja kamere po X osi
CameraMoveYSpeed	float	Hitrost premikanja kamere po Y osi
InvertCameraMoveX	bool	Obratno premikanje kamere po X osi
InvertCameraMoveY	bool	Obratno premikanje kamere po Y osi
CameraScrollSpeed	float	Hitrost približevanja kamere
InvertCameraScroll	bool	Obratno približevanja kamere
LoadLayers	bool	Vklop nalaganja nivojev
Preload	bool	Vklop prednalaganja nivojev
PreloadLayer	int	Do katerega nivoja naj se predpomni v primeru prednalaganja
MaxLoad	bool	Vklop minimalnega nivoja
MaxLoadLayer	int	Do katerega nivoja naj se maksimalno naloži v primeru omejevanja nalaganja
RapidLoading	bool	Vklop hitrega nalaganja - minimalne pohitritve za veliko slabšo uporabniško izkušnjo
LoadHeightmap	bool	Vklop nalaganja višinskih slik
LoadTexture	bool	Vklop nalaganja ortofoto posnetkov
SaveNewTextureToFile	bool	Vklop lokalnega predpomnjenja ortofoto posnetkov
SaveConverted NewTextureToRaw	bool	Vklop pretvarjanja predpomnjenih ortofoto posnetkov v format RAW

TestTotalLoadingTime	bool	Vklop razhroščevalne funkcije za testiranje časa nalaganja plošče - deluje le v Editorju
TestHeightmapLoadingTime	bool	Vklop razhroščevalne funkcije za testiranje časa nalaganja višinske slike - deluje le v Editorju
TestTextureLoadingTime	bool	Vklop razhroščevalne funkcije za testiranje časa nalaganja ortofoto posnetkov - deluje le v Editorju
DeloadChildren	bool	Vklop brisanja že naloženih plošč - če je izklopljeno, potem bodo vse plošče ostale v pomnilniku
DefaultStartSize	int	Osnovna velikost najvišje plošče
DefaultStart HeightmapResolution	int	Osnovna ločljivost višinskih slik
BaseFolder	string	Pot do višinskih slik - mape morajo biti med sabo ločene z dvema \namesto le enim
BaseTextureFolder	string	Pot do ortofoto posnetkov - mape morajo biti med sabo ločene z dvema \namesto le enim
RawFileEnding	string	Končnica RAW datotek
NumberOfLayers	int	Število nivojev
LevelFolders	string array	Imena nivojev ločenih z vejicami

LevelLoadHeight	int array	Višine pri katerih se začne nalagati nižji nivoji, vrednosti so ločene z vejicami
DefaultLayer	int	Številka začetnega nivoja
LevelsMaxXIgnore	int	Vrednost, potrebna za delovanje sistema - ne spreminjaj
IsDataStartingWithZero	int	Vrednost, potrebna za delovanje sistema - ne spreminjaj
TextureMinX	string	Začetna X koordinata sistema GK
TextureMinZ	string	Začetna Z koordinata sistema GK
TextureMaxX	string	Končna X koordinata sistema GK
TextureMaxZ	string	Končna Z koordinata sistema GK
TextureResolution	int	Ločljivost ortofoto posnetkov
UseLegacyAlignmentMatching	bool	Vklop starega sistema za poravnavo slik
TextureOffsetsX	float array	Odmiki po X osi za star sistem poravnave slik, vrednosti so ločene z vejicami
TextureOffsetsZ	float array	Odmiki po Z osi za star sistem poravnave slik, vrednosti so ločene z vejicami

Literatura

[1] Jordi Bonastre. Why should i use threads instead of coroutines? Dosegljivo: https://support.unity3d.com/hc/en-us/articles/208707516-Why-should-I-use-Threads-instead-of-Coroutines, 2016. [Dostopano: 10. 06. 2017].

[2] Alan Buis. Nasa, japan release improved topographic map of earth. Dosegljivo: https://www.nasa.gov/topics/earth/features/aster20111017.html, 2011. [Dostopano: 10. 06. 2017].

[3] Alan Buis. U.s. releases enhanced shuttle land elevation data. Dosegljivo: https://www.jpl.nasa.gov/news/news.php?release=2014-321, 2014. [Dostopano: 22. 06. 2017].

[4] Dempsey Caitlin. Mapping through the ages: The history of cartography. Dosegljivo: https://www.gislounge.com/mapping-through-the-ages/, 2011. [Dostopano: 10. 06. 2017].

[5] Center for Geospatial Analytics . Lidar airplane scanning. [Dostopano: 10. 06. 2017].

[6] Amit Chourasia, Steve Cutchin, and Brad Aagaard. Visualizing the ground motions of the 1906 san francisco earthquake. *Computers & Geosciences*, 34(12):1798–1805, 2008.

[7] Jackson Dunstan. Unity coroutine performance. Dosegljivo: http://jacksondunstan.com/articles/2981, 2015. [Dostopano: 10. 06. 2017].

[8] GIS Geography. How universal transverse mercator (utm) works. Dosegljivo: http://gisgeography.com/utm-universal-transverse-mercator-projection/, 2016. [Dostopano: 10. 06. 2017].

[9] AF Habib, J Kersting, TM McCaffrey, and AMY Jarvis. Integration of lidar and airborne imagery for realistic visualization of 3d urban environments. In *Proceedings of the International Society for Photogrammetry, Remote Sensing and Spatial Information Sciences,(ISPRS Congress)*, volume 37, pages 617–623, 2008.

[10] Karl-Heinz Kastner and Gerald Ostermayer. *Routing with Free Geodata on Mobile Devices*, pages 552–559. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[11] Gal Kos. 3d battlefield visualisation in unity. Diplomska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2016.

[12] Marko Kuder. Upodabljanje velikih količin letalskih podatkov lidar. Doktorska disertacija, Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru, 2014.

[13] Miha Lunar. Porazdeljeno sledenje žarkov za upodabljanje lasersko zajetih prostorskih podatkov. Diplomska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2016.

[14] Domen Lušina. Vizualizacija terena v unity3d. Seminarska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2017.

[15] Strzelecki Maksymilian. Coroutines in unity – encapsulating with promises [part 1]. Dosegljivo: http://blog.theknightsofunity.com/coroutines-unity-encapsulating-promises-part-1/, 2016. [Dostopano: 10. 06. 2017].

[16] Trevor Morris Photographics. The png file format. Dosegljivo: http://morris-photographics.com/photoshop/articles/png-format.html. [Dostopano: 10. 06. 2017].

[17] Mark Powell. Interactive 3d mars visualization. Dosegljivo: http://www.techbriefs.com/component/content/article/15256, 2012. [Dostopano: 10. 06. 2017].

[18] Stefan Tilkov. A brief introduction to rest. Dosegljivo: https://www.infoq.com/articles/rest-introduction, 2007. [Dostopano: 10. 06. 2017].

[19] John Wesolowski. How to plan optimizations with unity. Dosegljivo: https://software.intel.com/en-us/articles/how-to-plan-optimizations-with-unity, 2014. [Dostopano: 10. 06. 2017].

[20] David Whitehouse. Ice age star map discovered. Dosegljivo: http://news.bbc.co.uk/2/hi/science/nature/871930.stm, 2000. [Dostopano: 10. 06. 2017].

[21] Wikipedia, the free encyclopedia. Orthoperspective. [Dostopano: 10. 06. 2017].